Architecture

This tool implements a classic hybrid cryptosystem tailored for QR-friendly payloads. The GUI and CLI both route encryption through encrypt_data, which first compresses user input with zstd to minimize the footprint that must later be Base45-encoded.
Immediately after compression, encrypt_with_aes_gcm generates a fresh 256-bit session key and 96-bit IV, performs authenticated encryption, and returns the tuple of artifacts that the rest of the pipeline needs. This clean separation keeps all symmetric-crypto duties inside aes_module.py, including input validation, randomness, and error messaging.

Once the AES step succeeds, the RSA module loads the recipient's public key and OAEP-wraps the session key so only the intended private key holder can recover it. All transport-ready elements—RSA-wrapped key, IV, GCM tag, and ciphertext—are bundled into a typed CBOR envelope that captures protocol version, algorithm mix, and compression hint. Base45 encoding turns that binary blob into scanner-friendly text for QR codes or text files, so the GUI can display it verbatim or call qr_generator.py for visualization.

Decryption flows in the exact reverse order. decrypt_data accepts the Base45 string, decodes it, and relies on parse_envelope to enforce structural and metadata constraints. The RSA module unwraps the session key with OAEP/SHA-256, after which decrypt_with_aes_gcm authenticates and decrypts the ciphertext in one call. Only when the AES layer succeeds does the system invoke decompress_data, restoring the original bytes for text display or file output. Because each step guards its own inputs and raises precise errors, the GUI can show users whether a failure stemmed from a bad QR scan, a mismatched key, or tampered ciphertext.

The AES module's design choices underpin the system's guarantees. By centralizing key/nonce generation, validating sizes, and using PyCryptodome's high-level GCM API, it prevents misuse patterns (like IV reuse or truncated tags) that often plague ad-hoc AES code. The tuple return structure also minimizes redundant serialization work: the CBOR module and RSA wrapper receive exactly the bytes they need without reformatting, which keeps the envelope deterministic and makes future protocol versions straightforward to negotiate.